VCB-Studio教程16 Repair的用法与Contra-Sharp

本教程旨在讲述Repair的用法和简单扩展。

1. Repair的原理

Repair是根据RemoveGrain改变而来的，用法是Repair(filtered, source, mode=1)。

RemoveGrain总体的做法可以总结为：把中心像素，跟周边8个像素比较，让中心像素不要过高或者过低。比如说RemoveGrain(src, mode=1)，保证输出的每个像素，范围不超过其周边8个像素中最小值和最大值。

Repair的用法很类似，但是表现为，把flt的每个像素，跟src中对应部分的9个像素比较，让flt中的像素不要过高或者过低。flt通常为经过某种处理，可能引入突兀瑕疵的clip，而src是源。

比如说默认的Repair(flt,src,1)，会讲flt的每一个像素，跟src对应位置3x3像素比较。如果flt不超过src中9个像素的最大值或者最小值，那么不做处理，否则，会把flt的像素，替换为9个数中最大值或者最小值（取决于哪个更接近）。比如说9个数最小值是28，最大值是176，那么flt像素小于28的时候，会被改为28；在28-176之间则不变，大于176则会被改为176。

这是一种限制机制，保证flt的画面，跟src相比，不能太极端(跟源的领域比，出现过高或者过低的值)。所以Repair通常作为一种保护机制，任何可能引入突兀瑕疵的操作，可以用Repair做限制。

拓展开，Repair(flt, src, mode=k)，k=1/2/3/4的作用是flt的每一个像素，跟src对应位置3x3的比较，如果flt不超过第k大或者第k小的，那么保留；否则替换为第k大或者第k小的。k越大，越可能被限制，限制力度更高。

那么如果flt=src，执行Repair(flt,src,k)会怎样？不失一般性，不妨假设中心像素是偏大的。执行Repair(flt,src,1)是没有意义的；因为这时候flt=src, flt的每个像素，等于src的3x3中的中心像素。flt的任何像素永远不会被判定为过大或者过小；所以结果就是flt/src。

如果src的中心像素，在3x3的邻域内正好是第k大，那么不做调整，这时候比它大的有k-1个，这k-1个一定都在周边8个数字中，相当于中心像素不超过周边8个像素中第k-1大的；

如果src的中心像素，在3x3的邻域内是1…k-1大，那么需要调整到包括它自己在内，第k大的，或者说是周边8个像素中k-1大的，相当于中心像素被强制限制为周边8个像素中k-1大的。

这不偏不倚就是RemoveGrain(src,k-1)。

mode=k=1...4的时候，flt中心像素被限制在kth-Min和kth-Max之间。如果mode=10+k=11..14，逻辑是flt的中心像素被限制在MIN(kth-Min, src的中心像素)和MAX(kth-MAX, src的中心像素)之间。我们来分析下：

如果k=1，MIN(1st-Min, src的中心像素)= 1st-Min，MAX(1th-MAX, src的中心像素)= 1th-MAX，所以Repair(mode=11)和Repair(mode=1)是没有区别的；

如果k=2，MIN(2st-Min, src的中心像素)就不一定等于2st-Min了。有可能src的中心像素就是9个像素中最小的，这时候，Repair(mode=12)和Repair(mode=2)相比，上下限有可能更宽，当且仅当src的中心像素是最大/最小值。所以k=2/3/4的时候，如果src的中心像素本身很大或者很小，Repair(mode=10+k)比起Repair(mode=k)的限制力度是更弱的。

2. Repair的常见用法

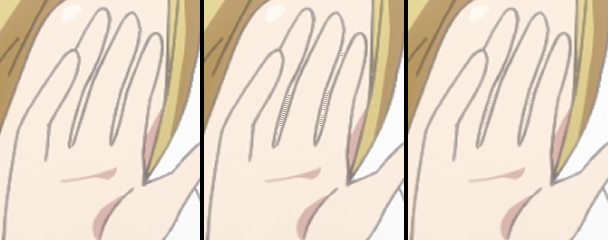
Repair被广泛用在可能引入突兀瑕疵的地方，比如说时域降噪，可能引入blending，把前后帧的信息加入本帧。这时候就可以用Repair来处理：

nr = src.MCTD()

Repair(nr, src, 2)

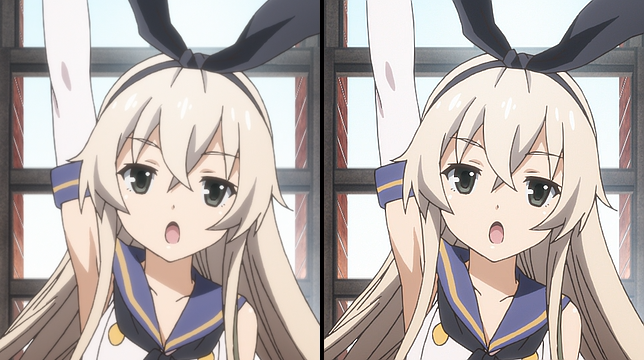
类似GSMC等时域降噪滤镜中，就有自带Repair的步骤。

又比如，SangNom做AA/Deint很可能引入瑕疵（左 vs 中），这时候通过Repair(aa, src, 3)则可以很好的限制这种瑕疵的强度：

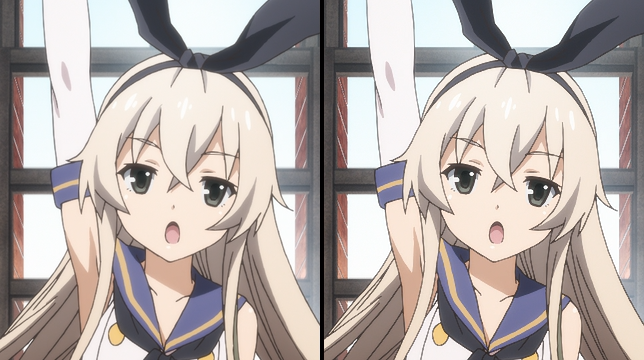


TAA的后处理中也有用Repair来修复瑕疵的做法。

3. Moderate Sharpening

之前咱们说了Unsharp Mask用来锐化：

这副效果有点太过分了。这时候我们就可以考虑用Repair(sharped, src, 1)来试着限制一下：



虽然不至于完全把瑕疵清理干净，但是比起不带限制，ringing应该基本全部去掉了；只剩下aliasing比较明显。

这就是RemoveGrain的doc中说的Moderate Sharpening，虽然还是很简单粗暴效果有待提高，但是起码提供了一个思路。

4. RGDering

上文中，我们从一个有ringing图像(sharp)，和一个没有ringing，但是清晰度低的图像(src)，运算出一个锐利度高但是没有ringing的图像(moderate sharpening)

写成函数就是：mdrt\_sharp = dering(sharp,src)

如果我们把第一个输入换成src，第二个输入换成src.blur呢？

blur = src.Minblur(r=1)

Repair(src, blur, mode=1)

这就是dering的基本手段：RGDering。它是removegrain的doc里自带的。注意它的用法是把src放在第一位，blur放在第二位。用于一个强ringing的图像（左）效果如下（右）：



效果还是很明显，虽然伴随着部分的线条虚化和residual ringing还是难以避免，不过这比直接轰blur破坏力小的多了。

这也解释了为啥之前Resizer(1)教程中的non-ringing resizer用法：

upscaled=core.fmtc.resample(src720p, 1920, 1080, kernel="lanczos",taps=4)

upscaled=core.rgvs.Repair(upscaled, core.fmtc.resample(src720p, 1920, 1080, kernel="gauss",a1=100),1)

本质上是用GaussResize做一个理论上不会出ringing的clip，然后用Repair去做一个RGDering。在实际使用中，可以通过调节a1(p)的值，这个值越小，GaussResize出来的结果越柔和，dering的强度越高。

5. Contra-Sharp/补偿性锐化

contrasharp是很常用的弥补手段，常常用于AA/deband/降噪/dering等柔化处理后，来补偿画面的锐度，强化被削弱的细节。比如下图是用Bilateral降噪的结果(core.bilateral.Bilateral(src16, sigmaS=1.5,sigmaR=0.015, algorithm=2))：



确实很有效的去掉了噪点，但是画面也被一定程度的柔化，一些弱线条（比如头发）损失较大。我们考虑对降噪后的画面做一个unsharpmask(11):



嗯……虽然画面锐利度回来了，柔化的线条救回来不少，但是，太锐利了，已经出现了过度锐化的痕迹。我们想让锐化后的锐利度不要超过源，该怎么办呢？

一个想法是，锐化的强度diff，不要超过src-nr的值，也就是降噪所损失的强度。即对于任何像素(无视符号正负)：

diff <= src-nr

上述可以用表达式求值实现；也可以近似用Repair实现：diff <= src-nr <= src-nr的3x3像素中最大值。

即，原图是

ABC

DEF

GHI

降噪后是

abc

def

ghi

我们希望中心像素的锐化强度，<= E-e

这样锐化后的值 <= e+E-e = E

我们考虑一个放大的近似，中心像素锐化强度<=max(A-a, B-b, ……I-i)

怎么实现呢?如下

nr16 = core.bilateral.Bilateral(src16, sigmaS=1.5,sigmaR=0.015, algorithm=2)

noise = core.std.MakeDiff(src16,nr16)

blur = core.rgvs.RemoveGrain(nr16, 11)

diff = core.std.MakeDiff(nr16,blur)

diff = core.rgvs.Repair(diff,noise,1)

res = core.std.MergeDiff(nr16,diff)

结果如下：



效果就比之前的好不少了。

Contra-Sharp被广泛用在不同场合，来对处理后的视频，相对于源做一个补偿性锐化。以上就是它最简单的实现方式；这个实现方式并非严格的保证锐化程度不超过源，而是用了一个近似，允许一定程度放大锐化强度（原则上锐化强度不超过自身损失，现在是允许不超过周边3x3像素的损失，即locally restricted而非pixel-wise restricted），这在目视效果上一般是很讨喜的。

不过如上实现可能存在的问题是：当9个像素的损失，全部大于锐化强度的时候，锐化强度会被放大到它们当中的最小者。等于说这时候锐化强度会被放大。这个问题的解决方案是：

diff = nr-nr.blur()

sharpdiff = Repair(diff,noise,1)

sharpdiff = min(sharpdiff, diff) #保证repair后的强度，不超过原来打算做锐化的强度

return nr+sharpdiff

这点我们会在表达式求值中继续说。

有另一种写法是这么写的：

diff = nr-nr.blur()

sharp = nr+diff

return Repair(sharp,src,1) #相当于基于nr，用src做限制，做一个Moderate Sharpening

参照上文说明，这种做法，锐化结果是被限制为max(A,B…I)，锐化强度，是被限制在max(A-e, B-e, ……I-e)，即把中心像素当做降噪后值，来算邻域每个像素的降噪损失。

这样的做法，可以视作降噪本身已经把3x3的邻域给抹平了，全都抹成了中心像素。显然，相对于之前的做法，这样做是放大降噪损失，从而削弱限制力度的。锐化出来的图像一般比起Contra-Sharp的更锐利，并且带有Moderate Sharpening那种锯齿风格。